Predicting Daily Land Average Temperature

Load the Data

Load the daily land-surface average anomaly data provided by Berkeley Earth collected from 1880 to 2022.

In [138…

```
import pandas as pd

url = "https://berkeley-earth-temperature.s3.us-west-1.amazonaws.com/Global/

'''
read the data from the url link
ignore the comments starting with '%'
ignore the header in the comments and assign manually
'''
df = pd.read_csv(url, sep=r"\s+", comment="%", header=None)

# assign column headers
column_names = ["Date Number", "Year", "Month", "Day", "Day of Year", "Anoma
df.columns = column_names

# df.to_csv("../data/raw.csv", index=False)
```

In [139…

```
df
```

Out[139…

	Date Number	Year	Month	Day	Day of Year	Anomaly
0	1880.001	1880	1	1	1	-0.692
1	1880.004	1880	1	2	2	-0.592
2	1880.007	1880	1	3	3	-0.673
3	1880.010	1880	1	4	4	-0.615
4	1880.012	1880	1	5	5	-0.681
...	...	...	...	...	...	...
52072	2022.568	2022	7	27	208	1.639
52073	2022.571	2022	7	28	209	1.631
52074	2022.574	2022	7	29	210	1.574
52075	2022.577	2022	7	30	211	1.577
52076	2022.579	2022	7	31	212	1.629

52077 rows × 6 columns

Data Preprocessing

In [140…
```
df.isna().sum()
```

Out[140…
```
Date Number    0
Year           0
Month          0
Day            0
Day of Year    0
Anomaly        0
dtype: int64
```

In [141…
```
df.dtypes
```

Out[141…
```
Date Number    float64
Year             int64
Month            int64
Day              int64
Day of Year      int64
Anomaly        float64
dtype: object
```

In [142…
```
# df = df.drop(columns=['Date Number'])
```

In [143…
```
BASELINE_TEMP = 8.59  # Jan 1951–Dec 1980 land-average temperature in celsiu

df['Temperature'] = df['Anomaly'] + BASELINE_TEMP
```

In [144…
```
month_dict = {
    1: 'January',
    2: 'February',
    3: 'March',
    4: 'April',
    5: 'May',
    6: 'June',
    7: 'July',
    8: 'August',
    9: 'September',
    10: 'October',
    11: 'November',
    12: 'December'
}

df['Month_Name'] = df['Month'].map(month_dict)
```

In [145…
```
df
```

Out [145…

	Date Number	Year	Month	Day	Day of Year	Anomaly	Temperature	Month_Name
0	1880.001	1880	1	1	1	-0.692	7.898	January
1	1880.004	1880	1	2	2	-0.592	7.998	January
2	1880.007	1880	1	3	3	-0.673	7.917	January
3	1880.010	1880	1	4	4	-0.615	7.975	January
4	1880.012	1880	1	5	5	-0.681	7.909	January
...	...	...	...	...	...	...	...	...
52072	2022.568	2022	7	27	208	1.639	10.229	July
52073	2022.571	2022	7	28	209	1.631	10.221	July
52074	2022.574	2022	7	29	210	1.574	10.164	July
52075	2022.577	2022	7	30	211	1.577	10.167	July
52076	2022.579	2022	7	31	212	1.629	10.219	July

52077 rows × 8 columns

Exploratory Data Analysis (EDA)

The goal of this EDA is to determine what type of predictive model will be the best fit for the data. A suitable regression method is to explored, as the target (global average land temperature) is continuous. After preprocessing the dataset, no presence of null values were found requiring attention. In order to make the Month_Name (converting from Month) feature more readable during EDA, the feature was converted to an ordinal feature, with Jan as the first in the order and December as the last in the order. Monthly trends in the data were considered over the years, however overall trends were found irrespective of the month the data was collected in. A cutoff year was decided for splitting the data into test and training data sets (see code below), and EDA was carried out on only the training portion of the data set to avoid violating the golden rule and double dipping. A randomized test split was not implemented as the model is desired for predicting temperatures in the future, so test data taken from the latest measurements represents the best evaluation of the regression model. Once the dataset was split, EDA was performed with several visualization strategies, the most informative of which are presented below. Discussions of the findings and rationale are found below as well.

In [146…

```
# Quick view of data and columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52077 entries, 0 to 52076
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Date Number  52077 non-null  float64
 1   Year         52077 non-null  int64  
 2   Month        52077 non-null  int64  
 3   Day          52077 non-null  int64  
 4   Day of Year  52077 non-null  int64  
 5   Anomaly      52077 non-null  float64
 6   Temperature  52077 non-null  float64
 7   Month_Name   52077 non-null  object 
dtypes: float64(3), int64(4), object(1)
memory usage: 3.2+ MB
```

In [147…

```
# Overview of statistics
df.describe()
```

Out[147…

	Date Number	Year	Month	Day	Day of Year	
count	52077.000000	52077.000000	52077.000000	52077.000000	52077.000000	5207
mean	1951.290840	1950.791693	6.512779	15.729228	182.688577	(
std	41.160006	41.160470	3.447762	8.799991	105.302088	
min	1880.001000	1880.000000	1.000000	1.000000	1.000000	-
25%	1915.648000	1915.000000	4.000000	8.000000	92.000000	-(
50%	1951.292000	1951.000000	7.000000	16.000000	183.000000	
75%	1986.936000	1986.000000	10.000000	23.000000	274.000000	(
max	2022.579000	2022.000000	12.000000	31.000000	365.000000	:

In [148…

```
# Verify no null values
df.isna().any()
```

Out[148…

```
Date Number    False
Year           False
Month          False
Day            False
Day of Year    False
Anomaly        False
Temperature    False
Month_Name     False
dtype: bool
```

In [149…

```
# month order from above dict values
month_order = list(month_dict.values())
month_order
```

Out[149… ['January',
 'February',
 'March',
 'April',
 'May',
 'June',
 'July',
 'August',
 'September',
 'October',
 'November',
 'December']

In [150…
```
# Order the month names as categorical
df['Month_Name'] = pd.Categorical(df['Month_Name'], categories=month_order,
df['Month_Name']
```

Out[150…
```
0        January
1        January
2        January
3        January
4        January
          ...
52072       July
52073       July
52074       July
52075       July
52076       July
Name: Month_Name, Length: 52077, dtype: category
Categories (12, object): ['January' < 'February' < 'March' < 'April' ... 'S
eptember' < 'October' < 'November' < 'December']
```

In [151…
```
# Split into test and train dataframes based on decided cutoff
# Beyond which the model will be scored and used for prediction
cutoff = 2012
test_df = df[df['Year'] > cutoff]
test_df
```

Out [151…

	Date Number	Year	Month	Day	Day of Year	Anomaly	Temperature	Month_Name
48578	2013.001	2013	1	1	1	0.489	9.079	January
48579	2013.004	2013	1	2	2	0.429	9.019	January
48580	2013.007	2013	1	3	3	0.572	9.162	January
48581	2013.010	2013	1	4	4	0.816	9.406	January
48582	2013.012	2013	1	5	5	0.769	9.359	January
...	...	...	...	...	...	...	...	...
52072	2022.568	2022	7	27	208	1.639	10.229	July
52073	2022.571	2022	7	28	209	1.631	10.221	July
52074	2022.574	2022	7	29	210	1.574	10.164	July
52075	2022.577	2022	7	30	211	1.577	10.167	July
52076	2022.579	2022	7	31	212	1.629	10.219	July

3499 rows × 8 columns

In [152…

```
# Get training data based on decided cutoff
train_df = df[df['Year'] <= cutoff]
train_df
```

Out [152…

	Date Number	Year	Month	Day	Day of Year	Anomaly	Temperature	Month_Name
0	1880.001	1880	1	1	1	-0.692	7.898	January
1	1880.004	1880	1	2	2	-0.592	7.998	January
2	1880.007	1880	1	3	3	-0.673	7.917	January
3	1880.010	1880	1	4	4	-0.615	7.975	January
4	1880.012	1880	1	5	5	-0.681	7.909	January
...	...	...	...	...	...	...	...	...
48573	2012.988	2012	12	27	361	0.134	8.724	December
48574	2012.990	2012	12	28	362	0.371	8.961	December
48575	2012.993	2012	12	29	363	0.532	9.122	December
48576	2012.996	2012	12	30	364	0.594	9.184	December
48577	2012.999	2012	12	31	365	0.631	9.221	December

48578 rows × 8 columns

In [153…

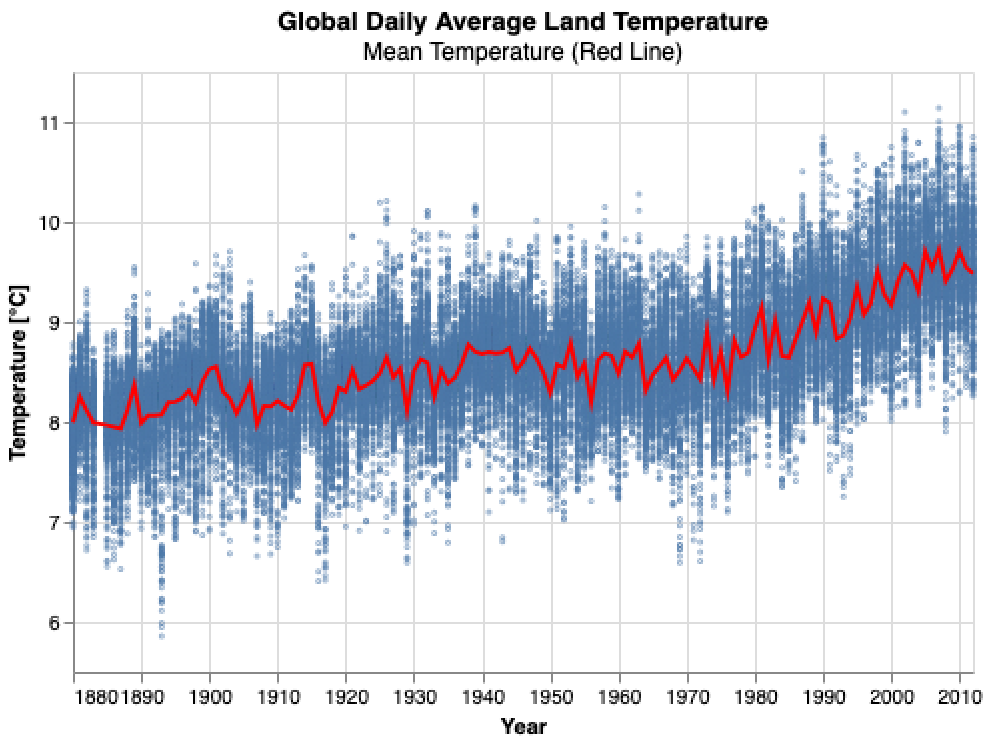
```
import altair as alt

# Simplify Working with Large Datasets
alt.data_transformers.enable('vegafusion')

# Ignore all warnings from the altair package for pdf rendering, warnings va
# resource for implementation:
# https://stackoverflow.com/questions/3920502/how-to-suppress-a-third-party-
import warnings
warnings.filterwarnings('ignore', module='altair')

# Configure Plot Sizes for pdf (D.R.Y)
plot_size = {'width': 450, 'height': 300}
facet_plot_size = {'width': 250, 'height': 200}
```

Due to the many measurements that occur in each year (contributing to plotting noise), the mean of the temperatures for each group of measurements taken in a year were plotted in addition to the raw data points of temperature over time. A general trend of increasing temperature over time with local fluctuations can be observed below.

In [154…

```
# Create scatter of raw data with some opacity to reduce plot noise
temp_points = alt.Chart(train_df,
        title=alt.Title(
        text='Global Daily Average Land Temperature',
        subtitle='Mean Temperature (Red Line)')
        ).mark_point(opacity=0.6, size=1).encode(
    x = 'Year:T',
    y = 'Temperature:Q')

# Create line plot of the mean of all the measurements in a given year
temp_line_mean = temp_points.mark_line(size=2, color='red').encode(
    x = alt.X('Year:T', title='Year'),
    y = alt.Y('mean(Temperature):Q', title='Temperature [°C]'
              ).scale(zero=False)
).properties(**plot_size)

# show the raw data distribution along with mean by year
temp_points+temp_line_mean
```

Out [154…

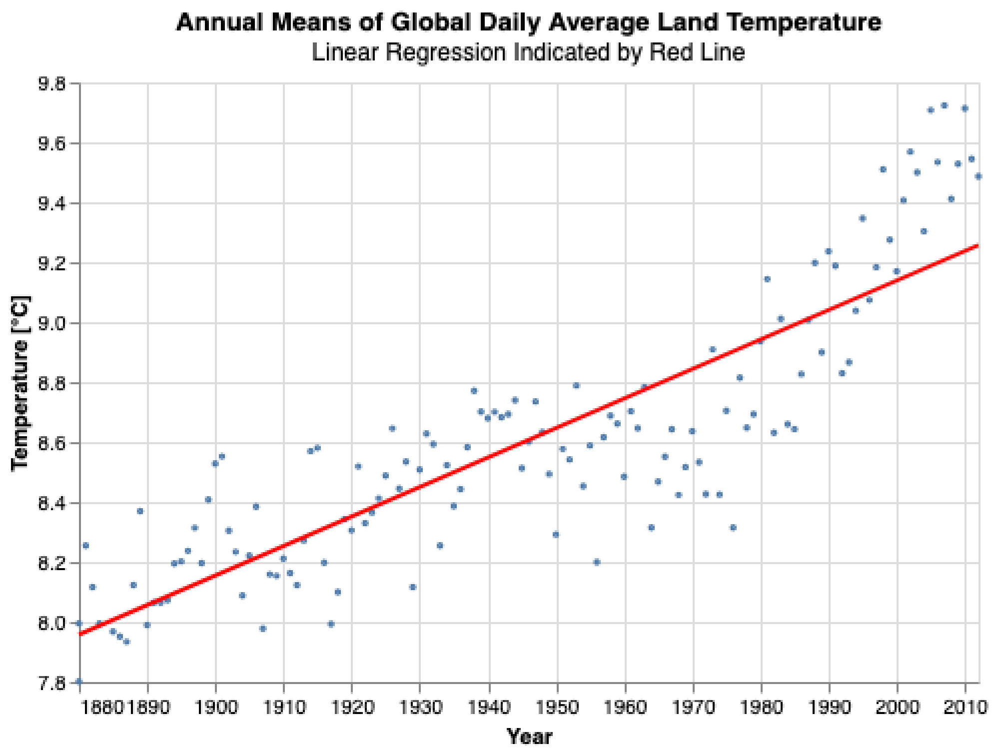


Using the mean data points by year, a linear regression model was fit to the training data to assess the viability of this approach in prediction. The linear regression fit can be seen to follow the overall trend, but misses information about local fluctuations, seen by the points above and below the line. A linear model has potential to generalize for unseen year examples but may be too simple for this analysis and will not be precise in its predictions for every year. A 5-year rolling average was also explored, due to its ability to adapt the curve more effectively to local fluctuations. However, a model of this type may not extrapolate well for unseen examples to predict future temperatures.

In [155…

```
# PLot a scatter plot of the mean temperatures for each year
temp_points_avg = alt.Chart(train_df,
        title=alt.Title(
        text='Annual Means of Global Daily Average Land Temperature',
        subtitle='Linear Regression Indicated by Red Line')
    ).mark_point(size=2).encode(
    x = alt.X('Year:T', title='Year'),
    y = alt.Y('mean(Temperature):Q',
              title='Temperature [°C]').scale(zero=False)
)

# Add the regression line to the scatter plot and properties
reg = temp_points_avg+temp_points_avg.mark_line(
    size=2, color='red').transform_regression(
    'Year',
    'Temperature'
).properties(**plot_size)

# Show the regression plot
reg
```

Out [155…

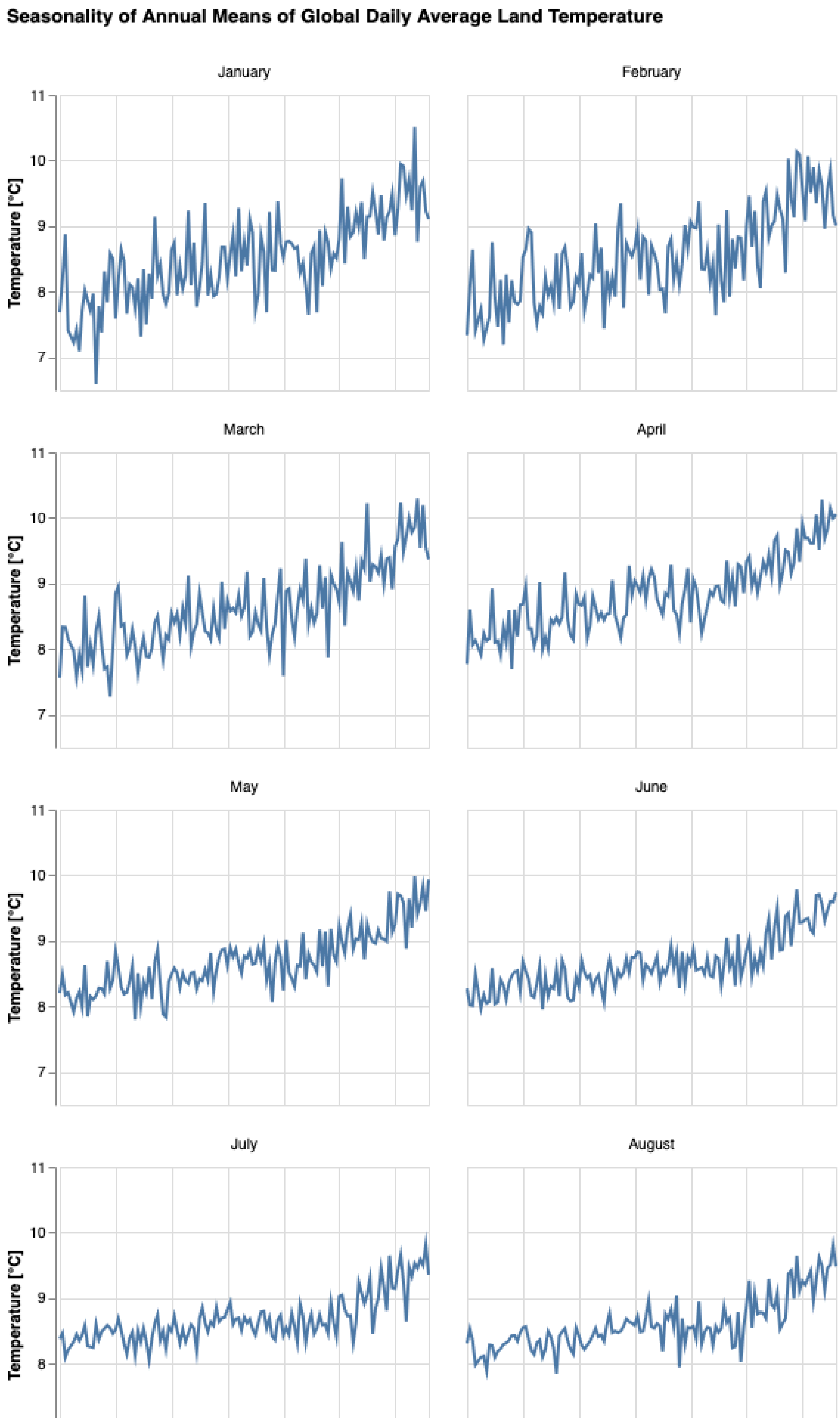


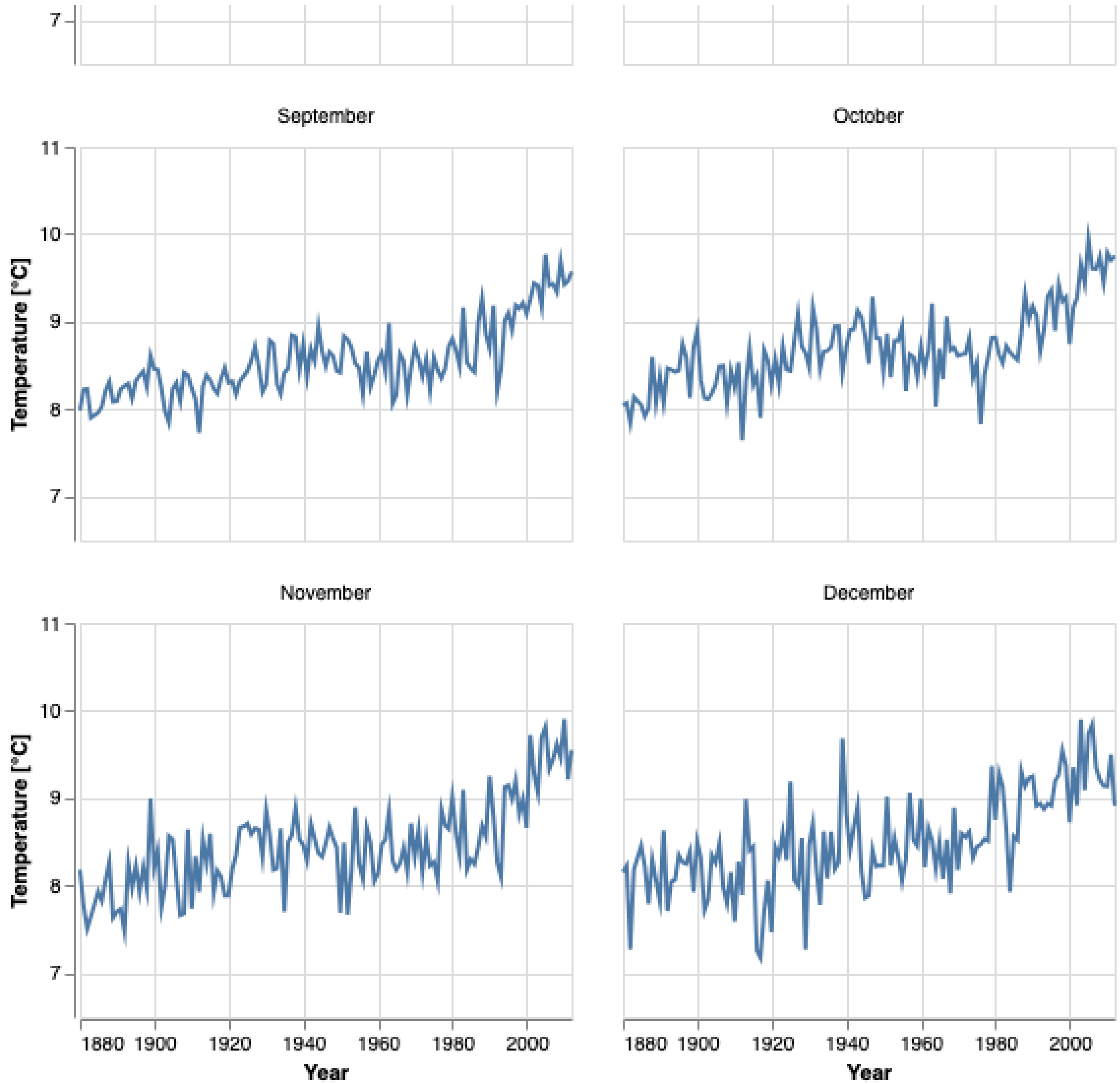
In order to check if the global average daily temperature over time was affected by seasonal phenomena, the annual means of these temperature measurements were analyzed by each month separated. It can be seen from the facet plot below that regardless of the season global daily average land temperature measurements are increasing non-trivially over time.

In [156…

```
# Average by year for each month in data
mean_per_month = train_df.groupby(
    ['Year','Month_Name'], observed=True)['Temperature'].mean().reset_index(

# For empty plotting title
mean_per_month[' ']=mean_per_month['Month_Name']

# Create Mean temperature plots and facet by month
temp_plot = alt.Chart(mean_per_month).mark_line().encode(
    x = 'Year:T',
    y = alt.Y('Temperature:Q', title='Temperature [°C]').scale(zero=False)
).properties(**facet_plot_size).facet(' ', columns=2)

# Show the plot with overall title
final_figure = alt.hconcat(temp_plot).properties(
    title='Seasonality of Annual Means of Global Daily Average Land Temperat
final_figure
```

Out [156…


Seasonality of Annual Means of Global Daily Average Land Temperature
January
February
March
April
May
June
July
August
Temperature [°C]
11
10
9
8
7



Finally, the distribution of the training data for temperature with respect to distinct year separated in by approximately 60 years were evaluated for distinct eras in the density plot below. The last year in the training data set (the cutoff year defined above) can be seen as significantly shifted to the right, indicating the increase in global daily average land temperature across time. Three distinct peaks in the density plot below reinforces the hypothesis that even when account for the variance in temperature due to local fluctuations, the overall trend of global temperature increasing over time remains.

In [157…

```
# Years to be analyzed separated by ~60 years
years_selection = [1880, 1960, cutoff]

# Select necessary data only
data_subset = train_df[train_df['Year'].isin(years_selection)]

# Create density plot for each selected year and add opacity to view overlap
temp_density  = alt.Chart(data_subset
    ).transform_density(
    'Temperature',
    groupby=['Year'],
    as_=['Temperature', 'density']
```

```
).mark_area(opacity=0.6).encode(
    x=alt.X('Temperature',title='Temperature [°C]'),
    y=alt.Y('density:Q',title='Density').stack(False),
    color = 'Year:N'
).properties(**plot_size, title = 'Distributions of Global Daily Average Lar

# Display the density plots by select years
temp_density
```

Out[157…